USER'S MANUAL FOR

VECTFIT

<u>version 2.1</u> for Matlab

written by :

Bjørn Gustavsen
SINTEF Energy Research
N-7465 Trondheim
NORWAY

E-mail: bjorn.gustavsen@sintef.no
Fax: +47 73597250

Download site:

Last revised : 27.10.2005
Total number of pages: 30

1. INTRODUCTION

This guide describes version 2.1 of the Matlab function `vectfit` for calculating a rational approximation of a given frequency domain response. It is intended to replace the previous version 1.0 (2000) by introducing significant improvements, including

- Option for relaxation of the non-triviality constraint in the vector fitting algorithm [2]. This often gives faster and more reliable convergence, in particular when fitting noisy responses
- Option for using sparse computations, enabling solution of larger systems.
- Option for usage of Normal Equations, enabling faster computations

`vectfit` and VF are abbreviations for "Vector Fitting". A thorough description of the method and its characteristics is given in [1]. Vector Fitting has since its first introduction in 1999 become a widely applied tool for frequency domain identification of linear systems, thanks to its robust and efficient formulation, and enforcement of guaranteed stable poles. Applications include

- Simulation of electromagnetic transients in power systems: Wide band modelling of transmission lines/cables, power transformers, instrument transformers, and frequency dependent network equivalents (FDNE). Vector fitting is used for pole identification of the phase domain transmission line models in EMTDC (Manitoba HVDC Research Centre) and EMTP-RV (IREQ/Hydro-Quebec).
- Signal integrity simulation of microwave systems: Modelling of systems, devices and transmission lines. Vector fitting is used in the SIDEA software by Optimal Corporation, in the Analog Office software of Applied Wave Research Inc.
- Specialized areas: Shielding analysis (EMC), antenna design, Green's functions representation.

To look for specific applications, check the download site (item "References") at http://www.energy.sintef.no/Produkt/VECTFIT/index.asp, or better, do a GoogleScholar search from http://scholar.google.com/

The function to be fitted can be either a single frequency response, or a vector of frequency responses. In the latter case, all elements in the vector will be fitted using a common pole set. This vector fitting capability gives powerful means of identifying models of multi-terminal systems, via columnwise fitting or even simultaneous fitting of the entire matrix. The resulting model can be expressed in either state-space form or pole-residue form.

The vector fitting algorithm replaces an initial set of poles with an improved set of poles using a pole relocation method based on least squares (LS) approximation of two linear problems [1]. The new poles can be reused as initial poles in an iterative procedure. Convergence is normally achieved in 2-3 iterations. (A single call to vectfit gives a single iteration). This procedure does not suffer from the ill-conditioning problems encountered by polynomial-based fitting methods.

Note: The program code is in the public domain and can be freely used by anyone. We only ask that if the program code is used in a scientific work, or in a commercial program, then the method should be called *Vector Fitting*, and reference should be made to the following :
B. Gustavsen and A. Semlyen, "Rational approximation of frequency domain responses by vector fitting", IEEE Trans. Power Delivery, vol. 14, no. 3, July 1999, pp. 1052-1061.

2. PROGRAM FILES

The package VFIT2.zip contains the following 19 files:

vectfit2_userguide.pdf %This document
vectfit_paper.pdf　　%Paper describing vector fitting
Relaxed_VF.pdf　　%Paper describing relaxation of vector fitting

vectfit2.m　　%The fitting routine (vector fitting)
tri2full.m　　%Auxiliary routine (used in `ex4c.m`, `ex4d.m`)
ss2pr.m　　%Auxiliary routine (used in `ex4c.m`, `ex4d.m`)

ex1.m　　%Fitting a vector of 1 rational function
ex2.m　　%Fitting a vector of 2 rational functions
ex3.m　　%Fitting a single element (transformer measurement)
ex4a.m　　%Fitting a single column (network equivalent)
ex4b.m　　%Columnwise fitting (network equivalent)
ex4c.m　　%Matrix fitting (network equivalent)
ex4d.m　　%Matrix fitting with passivity "precautions" (network equivalent)
ex5.m　　%Fitting a vector of 5 elements (transmission line modeling)
ex6.m　　%Fitting a pure time delay

03pk10.txt　　%data file for ex3.m
fdne.txt　　%data file for ex4a.m – ex4d.m
h.txt　　%data file for ex5.m
w.txt　　%data file for ex5.txt

`ex1.m`, `ex2.m`, `ex3.m`, `ex4a.m`, `ex4b.m`, `ex4c.m`, `ex5.m` and `ex6.m` are Matlab examples described in this guide.

3. COMPUTING REQUIREMENTS

The computer code has been tested on Matlab v6.5.0, v7.0.1, and v7.1.0.

No toolboxes are needed.

4. ROUTINE DESCRIPTION

4.1 Purpose

`vectfit` approximates a frequency response $\mathbf{f}(s)$ (generally a vector) with a rational function, expressed in the form of a sum of partial fractions:

$$\mathbf{f}(s) \approx \sum_{m=1}^{N} \frac{\mathbf{c}_m}{s - a_m} + \mathbf{d} + s\mathbf{e} \tag{1}$$

where terms $\mathbf{d}$ and $\mathbf{e}$ are optional. $\mathbf{c}_m$ and a_m are the residues and poles, respectively. If $\mathbf{f}(s)$ has N_c elements, all bold quantities in (1) have dimension ($N_c \times 1$). All poles $\{a_m\}$ are stable (lie in the left half plane). The model as returned by vectfit is for convenience expressed as parameters of a state-space model, i.e.

$$\mathbf{f}(s) \approx \mathbf{C}(s\mathbf{I} - \mathbf{A})^{-1}\mathbf{b} + \mathbf{d} + s\mathbf{e} \tag{2}$$

The user can choose between a model with real-only parameters or a model with real and complex conjugate parameters, depending on input parameter VF.complex_ss.

	VF.complex_ss =1	VF.complex_ss =0
A	Diagonal, complex	Block-diagonal, real (one 2x2 block for each complex pair)
b	Column vector of 1's	Column vector of 1's, 2's and 0's
C	Full, complex	Full, real
d	Column vector, real	Column vector, real
e	Column vector, real	Column vector, real

Note:
It is assumed that $\mathbf{f}(s)$ is complex conjugate. Only positive frequency samples should be specified.

4.2 Limitations

- The initial poles must be distinct. Otherwise, the solution is not unique and failure results when using the normal equations option (VF.NE=1).
- The response to be fitted cannot have multiple poles.

4.3 Specification

```
function [SER,poles,rmserr,fit]=vectfit2(f,s,poles,weight,VF)
%
%     ==========================================================
%     =    Vector Fitting                                      =
%     =    Version 2.1                                         =
%     =    Last revised: 27.10.2005                            =
%     =    Bjorn Gustavsen                                     =
%     =    SINTEF Energy Research, N-7465 Trondheim, NORWAY    =
%     =    bjorn.gustavsen@sintef.no                           =
%     =    http://www.energy.sintef.no/Produkt/VECTFIT/index.asp =
%     ==========================================================
%
% PURPOSE : Approximate f(s) with a state-space model
%
%           f(s)=C*(s*I-A)^(-1)*B +D +s*E
%
%             where f(s) is a singe element or a vector of elements.
%           When f(s) is a vector, all elements become fitted with a common
%           pole set.
%
% INPUT :
%
% f(s) : function (vector) to be fitted.
%        dimension : (Nc,Ns)
%                      Nc : number of elements in vector
%                      Ns : number of frequency samples
%
% s : vector of frequency points [rad/sec]
%        dimension : (1,Ns)
%
% poles : vector of initial poles [rad/sec]
%        dimension : (1,N)
%
% weight: the rows in the system matrix are weighted using this array. Can be used
%        for achieving higher accuracy at desired frequency samples.
%        If no weighting is desired, use: weight=ones(1,Ns).
%
%        Two dimensions are allowed:
%          dimension : (1,Ns) --> Common weighting for all vector elements.
%          dimension : (Nc,Ns)--> Individual weighting for vector elements.
%
% VF.relax==1 --> Use relaxed nontriviality constraint
% VF.relax==0 --> Use nontriviality constraint of "standard" vector fitting
%
% VF.kill=0 --> unstable poles are kept unchanged
% VF.kill=1 --> unstable poles are deleted
% VF.kill=2 --> unstable poles are 'flipped' into the left half plane
%             (kill=2 is the recommended choice)
%
% VF.asymp=1 --> Fitting with D=0,  E=0
% VF.asymp=2 --> Fitting with D~=0, E=0
% VF.asymp=3 --> Fitting with D~=0, E~=0
%
% VF.spy1=1 --> Plotting, after pole identification (A)
%               figure(3): magnitude functions
%                 cyan trace  : (sigma*f)fit
%                 red trace   : (sigma)fit
%                 green trace : f*(sigma)fit - (sigma*f)fit
%
% VF.spy2=1 --> Plotting, after residue identification (C,D,E)
%               figure(1): magnitude functions
%               figure(2): phase angles
%
% VF.logx=1 --> Plotting using logarithmic absissa axis
%
% VF.logy=1 --> Plotting using logarithmic ordinate axis
%
% VF.errplot=1   --> Include deviation in magnitude plot
%
% VF.phaseplot=1 -->Show plot also for phase angle
%
%
% VF.skip_pole=1 --> The pole identification part is skipped, i.e (C,D,E)
%                    are identified using the initial poles (A) as final poles.
```

```
%
% VF.skip_res =1 --> The residue identification part is skipped, i.e. only the
%                    poles (A) are identified while C,D,E are returned as zero.
%
% VF.use_normal=1 -->Solving Least Squares (LS) systems using the Normal Equations.
%              =0 -->Solving LS systems using QR decomposition
%
% VF.use_sparse=1 -->Sparsity is used for formulating and solving LS system of the
%                    pole identification problem.
%              =0 -->Full arithmetic is used (not very useful).
%
% VF.cmplx_ss  =1 -->The returned state-space model has real and complex conjugate
%                    parameters. Output variable A is diagonal (and sparse).
%              =0 -->The returned state-space model has real parameters only.
%                    Output variable A is square with 2x2 blocks (and sparse).
%
% OUTPUT :
%
%     fit(s) = C*(s*I-(A)^(-1)*B +D +s.*E
%
% SER.A(N,N)    : A-matrix (sparse). If cmplx_ss==1: Diagonal and complex.
%                              Otherwise, square and real with 2x2 blocks.
%
% SER.B(N,1)    : B-matrix. If cmplx_ss=1: Column of 1's.
%                           If cmplx_ss=0: contains 0's, 1's and 2's)
% SER.C(Nc,N)   : C-matrix. If cmplx_ss=1: complex
%                           If cmplx_ss=0: real-only
% SERD.D(Nc,1)  : constant term (real). Is non-zero if asymp=2 or 3.
% SERE.E(Nc,1)  : proportional term (real). Is non-zero if asymp=3.
%
% poles(1,N)    : new poles
%
% rmserr(1) : root-mean-square error of approximation for f(s).
%                 (0 is returned if skip_res==1)
% fit(Nc,Ns): Rational approximation at samples. (0 is returned if
%             skip_res==1).
%
%
% APPROACH: The identification is calculated using Vector Fitting, see
% reference in text box below. A modification has been introduced in v2 which
% makes convergence substantially faster and more reliable than in v1, see
% B. Gustavsen, "Improving the pole relocating properties of Vector
% Fitting", IEEE Trans. Power Delivery, accepted.
%
%
%*******************************************************************************
% NOTE: This program is in the public domain and may be used by anyone. If the  *
%       program code (or a modified version) is used in a scientific work, or   *
%       in a commercial program, then reference should be made to the following:*
%       B. Gustavsen and A. Semlyen, "Rational approximation of frequency       *
%       domain responses by Vector Fitting", IEEE Trans. Power Delivery,        *
%       vol. 14, no. 3, pp. 1052-1061, July 1999.                               *
%*******************************************************************************
```

4.4 Specification of initial poles

For functions with resonance peaks, the initial poles should be complex conjugate with weak attenuation, with imaginary parts β covering the frequency range of interest. The weak attenuation assures that the LS problem to be solved has a well-conditioned system matrix, and the distribution of the pairs over the frequency range reduces the probability that poles need to be relocated a long distance (avoiding need for many iterations).

The pairs should be chosen as follows :

$$a_n = -\alpha + j\beta, \quad a_{n+1} = -\alpha - j\beta \tag{2}$$

where

$$\alpha = \beta/100 \tag{3}$$

Typically, β is specified to be linearly spaced over the frequency range of interest (recommended choice). In some instances, a logarithmic distribution gives faster convergence. The selection of initial poles is demonstrated in several of the example m-files. Code sections for producing initial poles would look like this (w is the frequency given in rad/sec).

```
N=20; %Order of approximation

%Complex conjugate pairs, linearly spaced:
bet=linspace(w(1),w(end),N/2);
poles=[];
for n=1:length(bet)
  alf=-bet(n)*1e-2;
  poles=[poles (alf-i*bet(n)) (alf+i*bet(n)) ];
end



% Complex conjugate pairs, logarithmically spaced :
bet=logspace(log10(w(1)),log10(w(end)),N/2);
poles=[];
for n=1:length(bet)
  alf=-bet(n)*1e-2;
  poles=[poles (alf-i*bet(n)) (alf+i*bet(n)) ];
end
```

When fitting very smooth functions, one may also use real initial poles, for instance:

```
%Real poles poles, logarithmically spaced :
poles=-2*pi*logspace(log10(w(1)),log10(w(end)),N);
```

4.5 Weighting

Weighting is a powerful way of controlling the accuracy of the resulting approximation. This is achieved via array weight. Two dimensions are permitted:
1. weight(Nc,Ns)
2. weight(1,Ns)

where
   Nc: number of elements to be fitted (= n.o. rows in array f)
   Ns: number of frequency samples (= n.o. columns in array f)

The first dimension option allows to specify independent weighting for the elements of f(s), while the second dimension option is intended for specifying a common weighting.

When modelling systems/devices that can interact with the remainder of the system, error magnifications can easily result when the impedance of the connected network is very different from the impedance used in the "measurement" (short circuit in admittance representation). The likelihood of large magnifications can be avoided by appropriate weighting, see Table 6.1.

Table 4.1 Some useful weighting schemes

Scheme	Independent weighting	Common weighting
1) No weight	weight=ones(Nc,Ns);	weight=ones(1,Ns);

2)
Strong
inverse
weight
	weight=1./(abs(f));	weight=zeros(1,Ns);
		for k=1:Ns
		  weight(1,k)=1/norm(f(:,k));
		end

3)
Weaker
inverse
weight
	weight=1./sqrt(abs(f));	weight=zeros(1,Ns);
		for k=1:Ns
		  weight(1,k)=1/sqrt(norm(f(:,k)));
		end

The inverse weighting schemes in Table 4.1 gives a strong weight on elements in **f**(s) where they are small. For instance, if **f**(s) is a scalar, weight scheme 2) in Table 4.1 results in a high weight where the magnitude of **f**(s) is small, thus tending to minimize the relative deviation instead of the absolute deviation. When fitting a vector, higher weight is also placed on small elements than on large elements.

One difficulty with using a the inverse weight is when **f**(s) contains noise since the noise content relative to the signal strength tends to be high where elements are small. It is therefore a chance that VF tries to fit the noise. This problem can be alleviated by using a weaker inverse weight, for instance scheme 3) in Table 4.1.

It is remarked that for pure transfer function modelling (such as the propagation function in transmission line modelling), one should normally <u>not</u> use any weight (unless a certain frequency band is very important).

Also, note that the sampling effectively represents a weighting: Increasing the sampling density in some frequency band makes the LS solver place more weight on this band.

4.6 Iterations

To improve accuracy, `vectfit` may be called again with the new poles (kept in `poles`) used as (new) initial poles. This is typically done as follows

```
for iter =1:Niter
  [SER,poles,rmserr,fit]=vectfit2(f,s,poles,weight,VF)
end
```

4.7 Recommended parameter settings

Parameter	Recommended setting	Comment
VF.relax	1	Use vector fitting with relaxed nontriviality constraint
VF.kill	2	Enforces stable poles by pole flipping
VF.spy1	0	Will *not* plot the fitting result associated with the first stage in VF (pole identification).
VF.skip_pole	0	Will *not* skip the calculation of poles. Using "1" is useful when i) poles have been obtained by some other means/programs, ii) refitting tafter throing out high-frequency poles (see comment for `ex4d.m`)
VF.skip_res	0	Will *not* skip the calculation of **C**,**D**,**E**. Using "1" can be useful for increasing computation speed, see explanation in Section 4.8)
VF.use_normal	1	Will use Normal Equations, which gives fast computations. Use "0" if ill-conditioning is a problem (will solve using QR decomp.).
VF.use_sparse	1	Sparse calculations are used for the pole identification part. Gives faster computations and reduced memory requirements when fitting vectors. When fitting pure scalars, using "0" may give slightly faster computations.

4.8 Ill-conditioning

In some situations the call to `vectfit2.m` may result in a warning message
```
Warning: Matrix is close to singular or badly scaled.
         Results may be inaccurate. RCOND = 1.059299e-017.
```

This essentially means that the solution is not well defined; it usually appears when fitting with very high orders so that the fitting error starts approaching machine precision. It can also happen when poles become very close.

Normally, this message has no practical consequences (use 'warning off' to get rid of it). An exception is when you are creating a lumped network equivalent as you may get some extremely large or extremely small circuit elements. In that case, one should consider to reduce the order.

4.9 Speed-issues

- When the number of iterations to be done by VF is known in advance, one can reduce the computation time by specifying `VF.skip_res=1`, which results in that the residue calculations are skipped. Make sure that `VF.res_skip` is set to 0 before the last iterations, for instance

```
VF.skip_res=1;
for iter=1:Niter
  if iter==Niter, VF.skip_res=0;end
  [SER,poles,rmserr,fit]=vectfit2(f,s,poles,weight,VF);
end
```

- With VF.spy2=1, plots are produced for the magnitude functions and the phase angles (when `VF.skip_res=0)`. The computation time can be reduced by skipping plotting, which is achieved by setting VF.spy2=0.

- With parameter VF.legend=1, the produced plots will include legends. Unfortunately, the inclusion of legends can be <u>very time consuming</u>. This is particularly a problem when fitting scalars using many iterations as the legends can dominate the computation time. This problem is avoided by setting VF.legend=0. Also, when producing plots during iterations, one should only include legends for the last iteration, for instance

```
VF.legend=0;
for iter=1:Niter
  if iter==Niter, VF.legend=1; end
  [SER,poles,rmserr,fit]=vectfit2(f,s,poles,weight,VF);
end
```

- When fitting long vectors, the computation time per iteration can become considerable. In such cases one can reduce the total computation time by calculating an improved set of initial poles. This can be achieved by computing a weighted (or

unweighted) sum of the elements in the vector. This is demonstrated in examples `ex4b.m–ex4d.m`.

4.10 Auxiliary routines

The following routines are included for convenience:

ss2pr.m

This routine converts a state-space model into a pole-residue model.
<u>Note</u>: it only works correctly when the state-space model has been fitted with a *common pole set*.

Example:
`[R,a]=ss2pr(A,B,C);`

R is a 3 dimensional vector holding the residue matrices, *a* is a column vector holding the poles. For a system with Nc terminals fitted with N common poles, the dimensions are:

```
A(Nc*N,Nc*N)
B(Nc*N,Nc)
C(Nc,Nc*N)

R(Nc,Nc,N)
A(N,1)
```

The pole-residue formulation is particularly useful when the model is to be realized in the form of a lumped network. Usage of the routine is demonstrated in example `ex4c.m`.

tri2full.m

This routine is useful when fitting symmetrical matrices (e.g. **Y**) using a common pole set. Because of symmetry, it is sufficient to fit only the lower triangle of **Y** (which saves computation time and memory over fitting the full matrix). The resulting state-space model can afterwards be modified to include the lower triangle by a call to `tri2full.m`.

`[SER]=tri2full(SER);`

Usage of the routine is demonstrated in example `ex4c.m`.

5. TUTORIAL EXAMPLE (ex1.m)

Consider the frequency response

$$f(s) = \frac{2}{s+5} + \frac{30 + j40}{s - (-100 + j500)} + \frac{30 - j40}{s - (-100 - j500)} + 0.5$$

Assume that we know $f(s)$ only as a discrete function. In this case $f(s)$ is rational, so a good fitter should certainly be able to find a very accurate approximation.

The following Matlab program does the job:

```
% ex1.m
%
% Fitting an artificially created frequency response (single element)
%
% -Creating a 3rd order frequency response f(s)
% -Fitting f(s) using vectfit2.m
%    -Initial poles: 3 logarithmically spaced real poles
%    -1 iteration
%
% This example script is part of the vector fitting package (v2.0)
% Last revised: 17.10.2005.
% Created by:   Bjorn Gustavsen.
%
clear all

%Frequency samples:
Ns=101;
s=2*pi*i*logspace(0,4,Ns);

disp('Creating frequency response f(s)...')
for k=1:Ns
  sk=s(k);
  f(1,k) = 2/(sk+5) + (30+j*40)/(sk-(-100+j*500)) ...
+ (30-j*40)/(sk-(-100-j*500)) + 0.5;
end

%Initial poles for Vector Fitting:
N=3; %order of approximation
poles=-2*pi*logspace(0,4,N); %Initial poles

weight=ones(1,Ns); %All frequency points are given equal weight

VF.relax=1;       %Use vector fitting with relaxed non-triviality constraint
VF.kill=2;        %Enforce stable poles
VF.asymp=3;       %Include both D, E in fitting
VF.skip_pole=0;   %Do NOT skip pole identification
VF.skip_res=0;    %Do NOT skip identification of residues (C,D,E)
VF.use_normal=1;  %Use Normal Equations instead of QR decomp.
VF.use_sparse=1;  %Use sparse computations (pole identification)
VF.cmplx_ss=1;    %Create complex state space model

VF.spy1=0;        %No plotting for first stage of vector fitting
VF.spy2=1;        %Create magnitude plot for fitting of f(s)
VF.logx=1;        %Use logarithmic abscissa axis
VF.logy=1;        %Use logarithmic ordinate axis
VF.errplot=1;     %Include deviation in magnitude plot
VF.phaseplot=1;   %Also produce plot of phase angle (in addition to
                  %magnitiude)
VF.legend=1;      %Do include legends in plots

disp('vector fitting...')
[SER,poles,rmserr,fit]=vectfit2rev(f,s,poles,weight,VF);
disp('Done.')
```

```
disp('Resulting state space model:')

A=full(SER.A)
B=SER.B
C=SER.C
D=SER.D
E=SER.E
rmserr
```

Executing the program gave the following results :

```
>> ex1
Creating frequency response f(s)...
vector fitting...
Done.
Resulting state space model:

A =
 -5.0000e+000                          0                          0
            0          -1.0000e+002 +5.0000e+002i                 0
            0                         0          -1.0000e+002 -5.0000e+002i


B =
     1
     1
     1


C =
   2.0000e+000            3.0000e+001 +4.0000e+001i   3.0000e+001 -4.0000e+001i


D =
   5.0000e-001


E =
   8.4466e-017


rmserr =
   4.8426e-011
```

Thus, the parameters have been accurately identified and so the fitting is nearly perfect (root-mean-square error smaller than $1E{-}10$).

In addition, two figures have been created on the screen:
- Figure(1) shows the magnitude of $f(s)$ (cyan) and of the approximation (dashed magenta), and the magnitude of the complex devation (green).
- Figure(2) shows the phase angle of $f(s)$ (cyan) and of the approximation (dashed magenta).

The two figures are shown on the next page.

You should try this yourself! Experiment with the plotting options (`VF.spy2`, `VF.logx`, `VF.logy`, `VF.errplot`, `VF.phaseplot`).
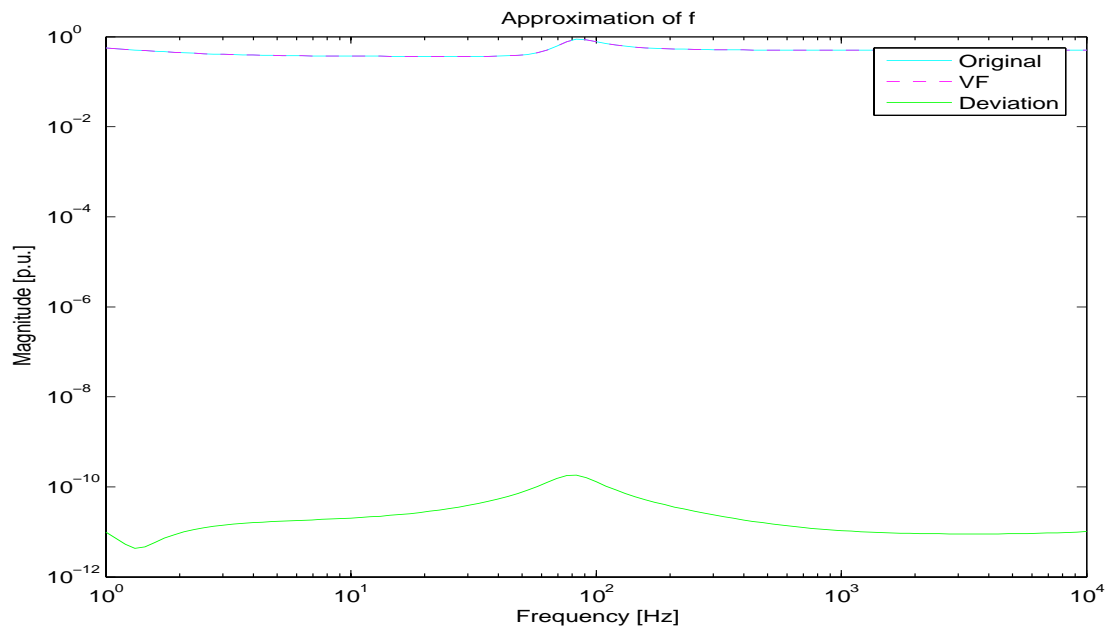
Fig. 5.1 Magnitude plot (figure(1))
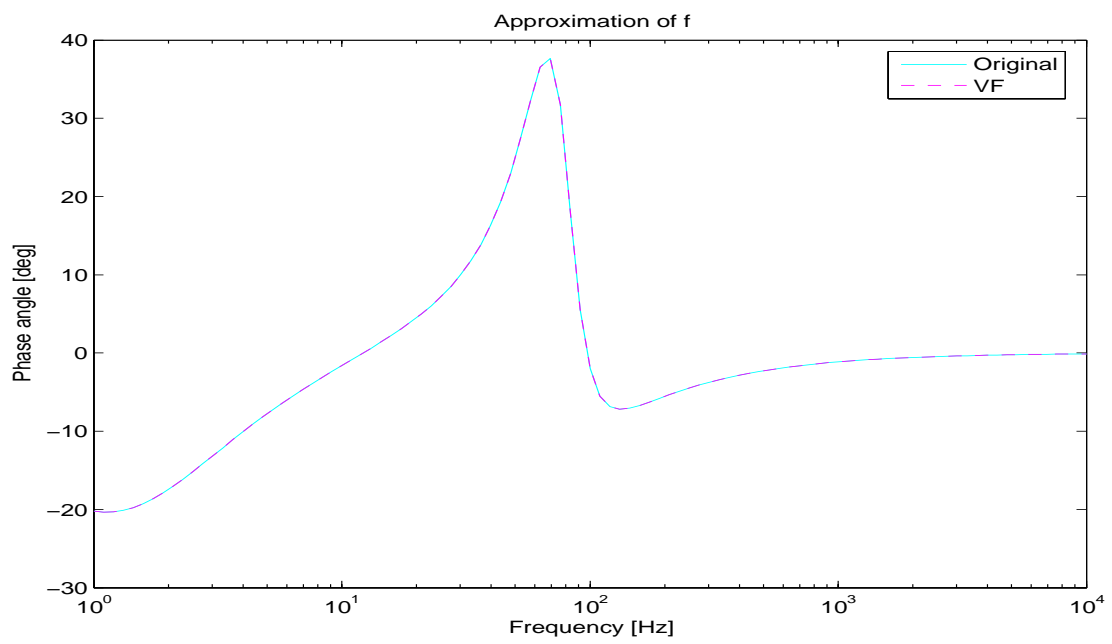
Fig. 5.2 Phanse angle plot (figure(2))

In the example (`ex1.m`), the parameter specification included
VF.complex_ss=1

Changing this value to "0" produces a state space model with real-only variables. With this
modification, running `ex1.m` will produce:

```
>> ex1
Creating frequency response f(s)...
vector fitting...
Done.
Resulting state space model:

A =
 -5.0000e+000               0               0
            0 -1.0000e+002   5.0000e+002
            0 -5.0000e+002 -1.0000e+002
B =
     1
     2
     0

C =
  2.0000e+000   3.0000e+001   4.0000e+001

D =
  5.0000e-001

E =
  8.4466e-017

rmserr =
  4.8426e-011
```

6. OTHER EXAMPLES

ex2.m – Fitting a vector of two rational elements

This program constructs an artificial scalar function $f(s)$ having two elements from predefined partial fractions.

- 18 complex starting poles are used
- `vectfit2` is called 3 times with the new poles used as starting poles
- Residue calculation is skipped, except for in the last (third) iteration
- A real-only state-space model is created

```
>> ex2
vector fitting...
   Iter 1
   Iter 2
   Iter 3

rms =
            0
            0
  9.0107e-012
```

ex3.m – Fitting a measured frequency response (transformer)

- The measured response of a transformer $f(s)$ (scalar) is read from file.
- 6 complex starting poles are used (3 complex pairs)
- vectfit2 is called 5 times with the new poles reused as starting poles

Running ex3.m gives the result in Fig. 6.1.
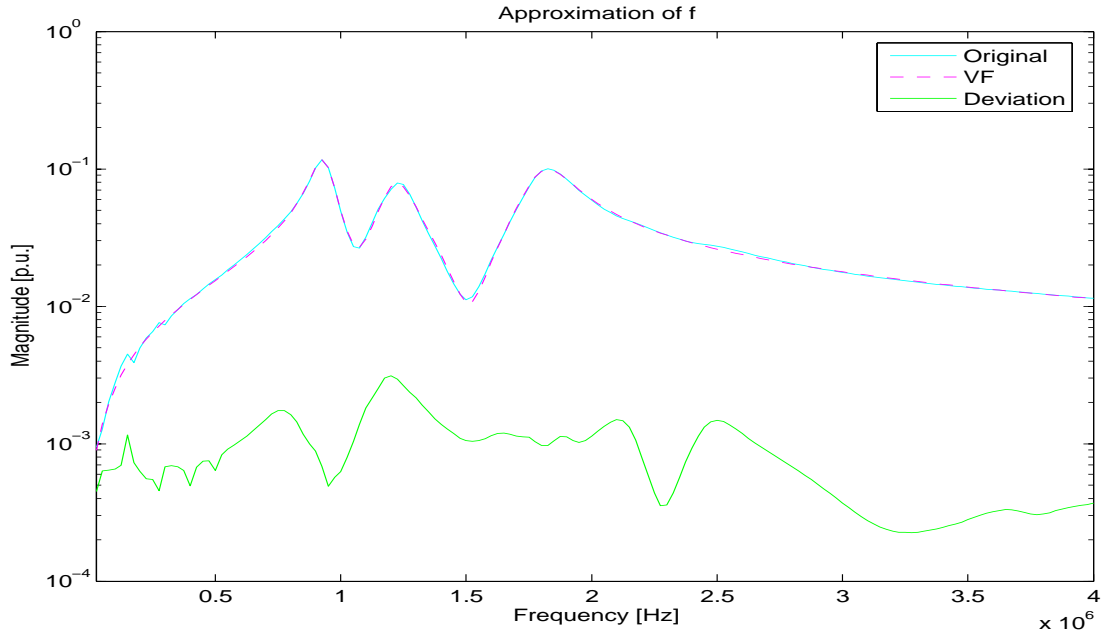
Fig. 6.1 Magnitude function (*N*=6, no weight)

Try increasing the order of the approximation to 30 (put *N*=30 on line #38). Also, try deleting the comment character '%' on line #49. The latter has the effect that $f(s)$ becomes weighted with the inverse of its magnitude. This is seen to cause the deviation to become small where the magnitude is small, meaning that we are minimizing the *relative* deviation instead of the *absolute* deviation. The effect on the magnitude function is shown in Fig. 6.2.
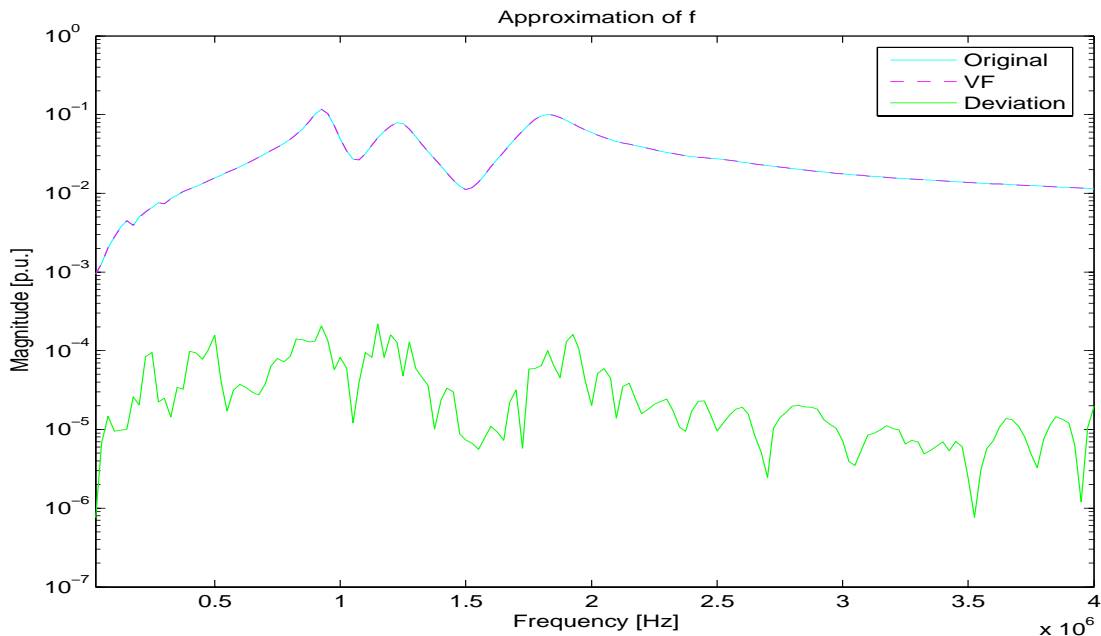
Fig. 6.2 Magnitude function (N=30, weighting with inverse of magnitude)

ex4a.m –Fitting a single column (Network equivalent)

This program reads from file a calculated terminal admittance matrix **Y** of a power system distribution system (`fdne.txt`). The distribution system has two 3-phase buses as terminals (A, B). The 6×6 admittance matrix **Y** with respect to these terminals has been calculated in the frequency range 10 Hz – 100 kHz.
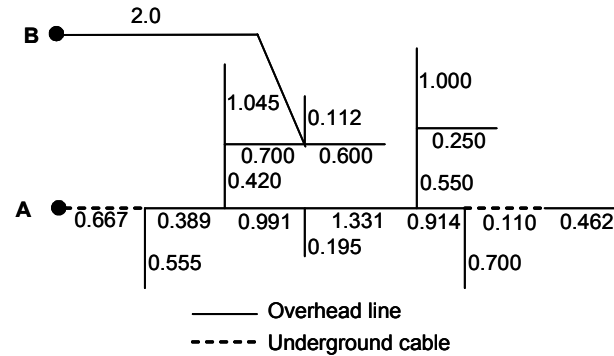
Fig. 6.3 Power system distribution system

`ex4a.m` fits the first column of **Y** as follows
- **f**(s) contains Nc=6 elements
- Ns=300 frequency samples
- N=50 initial poles (complex pairs)
- 5 iterations for the fitting of **f**(s)
- Weighting with inverse of the square root of the magnitude of **f**(s)
- Plots are updated in each iteration, legends are added after final iteration

The command window dialogue:

```
>> ex4a
Reading data from file ...
-----------------S T A R T-------------------------
*****Fitting column...
    Iter 1
    Iter 2
    Iter 3
    Iter 4
    Iter 5
-------------------E N D----------------------------
Elapsed time is 6.540000 seconds.
>>
```

The produced plots (magnitude, phase angle)
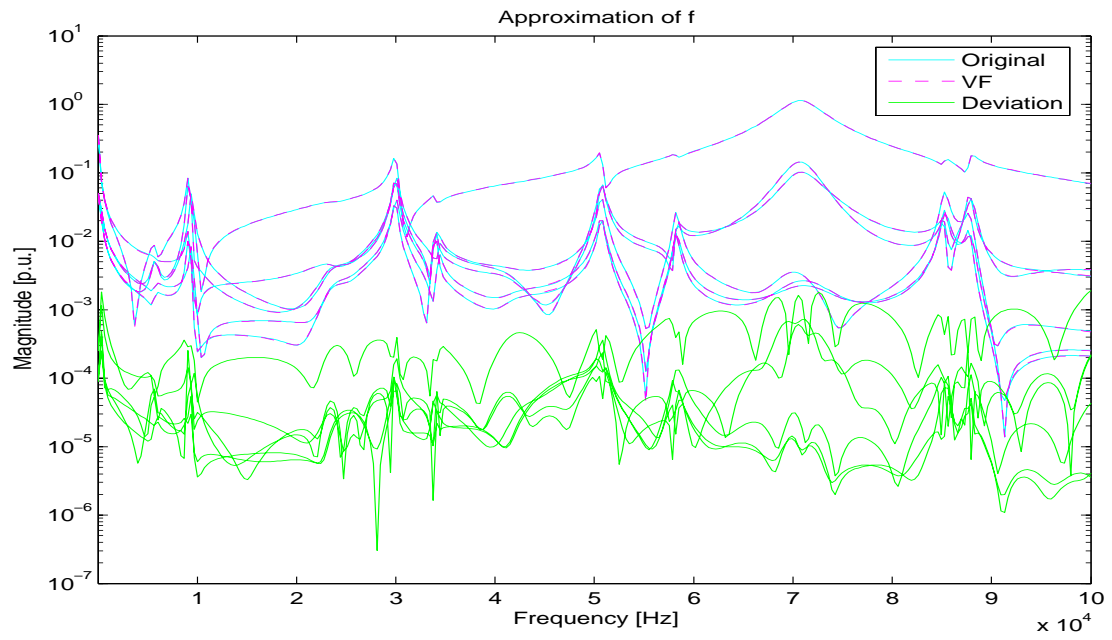
Fig. 6.4 Magnitude plot
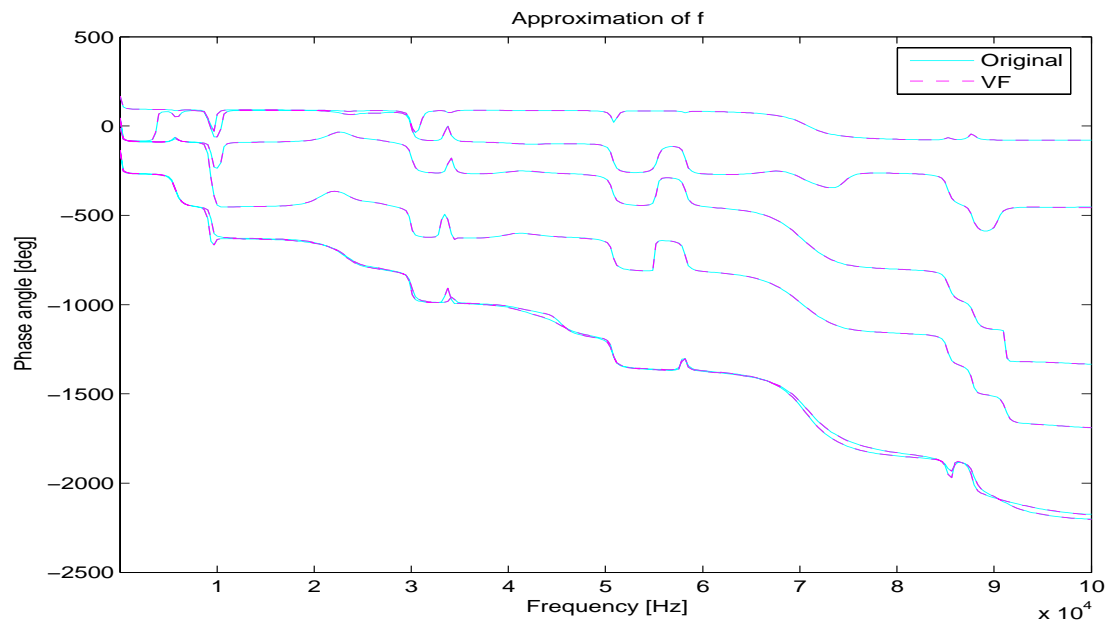
Fig. 6.5 Phase angle plot

ex4b.m – Columnwise fitting, improving initial poles

We now calculate a complete state-space model by fitting **Y** column-by-column. The state space model of the different columns are combined into a complete model.

Excerpt from code:

```
for col=1:N

  [A,B,C,D,E,rmserr,fit]=vectfit2(f,s,A,weight,VF);

  %Stacking the column contribution into complete state space model:
  AA=blkdiag(AA,A);
  BB=blkdiag(BB,B);
  CC=[CC C];
  DD=[DD D];
  EE=[EE E];

end
```

For a three-terminal system the result would have looked as follows:

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_1 & 0 & 0 \\ 0 & \mathbf{A}_2 & 0 \\ 0 & 0 & \mathbf{A}_3 \end{bmatrix}, \ \mathbf{B} = \begin{bmatrix} \mathbf{b}_1 & 0 & 0 \\ 0 & \mathbf{b}_2 & 0 \\ 0 & 0 & \mathbf{b}_3 \end{bmatrix}, \ \mathbf{C} = \begin{bmatrix} \mathbf{C}_1 & \mathbf{C}_2 & \mathbf{C}_3 \end{bmatrix},$$

$$\mathbf{D} = \begin{bmatrix} \mathbf{d}_1 & \mathbf{d}_2 & \mathbf{d}_3 \end{bmatrix}, \ \mathbf{E} = \begin{bmatrix} \mathbf{e}_1 & \mathbf{e}_2 & \mathbf{e}_3 \end{bmatrix}$$

- 50 initial poles (complex pairs)
- The initial poles are improved on by fitting the (weighted) column sum of the first column (5 iterations), before fitting the columns
- The columns are fitted independently (3 iterations)
- Weighting with inverse of the square root of the magnitude of **f**(s)
- For each column, residue calculation is skipped except for the last iteration
- At end of program, the result for all columns are gathered in single plot (figure(3))

The command window dialogue:
```
>> ex4b
Reading data from file ...
-----------------S T A R T-------------------------
****Improving initial poles by fitting column sum (1st column)...
   Iter 1
   Iter 2
   Iter 3
   Iter 4
   Iter 5
****Fitting column #1 ...
   Iter 1
```

```
    Iter 2
    Iter 3
****Fitting column #2 ...
    Iter 1
    Iter 2
    Iter 3
****Fitting column #3 ...
    Iter 1
    Iter 2
    Iter 3
****Fitting column #4 ...
    Iter 1
    Iter 2
    Iter 3
****Fitting column #5 ...
    Iter 1
    Iter 2
    Iter 3
****Fitting column #6 ...
    Iter 1
    Iter 2
    Iter 3
-------------------E N D----------------------------
Elapsed time is 18.658000 seconds.
>>
```

During calculations, a magnitude plot is produced for the fitting of each column. At the end of the program, a magnitude plot for all columns is produced:
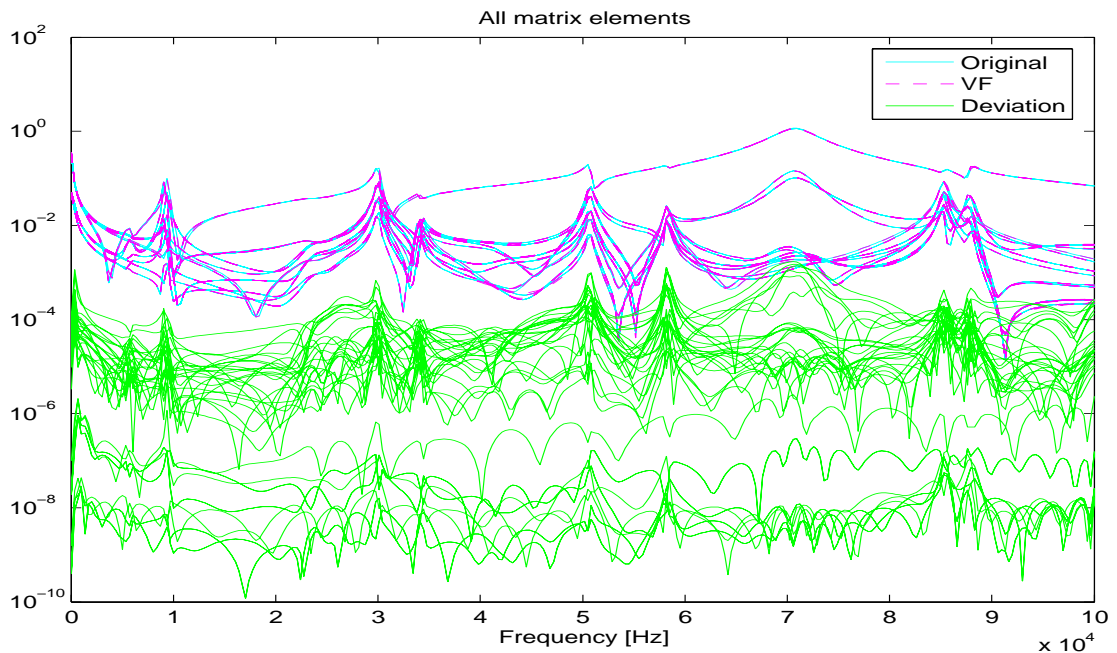
Fig. 6.6 Magnitude plot (figure(3))

ex4c.m – Matrix fitting

In this example, all elements of **Y** are fitted with a common pole set. The computations produce a state space model (A,B,C,D,E) plus a pole-residue model (a,R,D,E).

- The elements of the lower triangle of **Y**(which is symmetric) are stacked are stacked into a single vector **f**(s). **f**(s) has 21 elements.
- 50 initial poles (complex pairs)
- The initial poles are improved on by fitting the (weighted) column sum (5 iterations)
- The column (**f**) is fitted using 3 iterations
- Weighting with inverse of the square root of the magnitude of **f**(s)
- The state space model (lower triangle) is converted into a state space model for the full **Y** (using auxiliary routine `tri2full.m`)
- The state space model is converted into a pole-residue model (using auxiliary routine `ss2pr.m`)

The command window dialogue:
```
>> ex4c
Reading data from file ...
----------------S T A R T------------------------
****Stacking matrix elements (lower triangle) into single column....
****Calculating improved initial poles by fitting column sum ...
    Iter 1
    Iter 2
    Iter 3
    Iter 4
    Iter 5
****Fitting column ...
    Iter 1
    Iter 2
    Iter 3
****Transforming model of lower matrix triangle into state-space
model of full matrix....
****Generating pole-residue model....
-------------------E N D----------------------------
Elapsed time is 14.432000 seconds.
>>
```

The resulting magnitude plot is shown in fig. 6.7. The accuracy is lower than in fig. 6.6, because fitting with a common pole set is more constrained than using a separate pole set for each column.
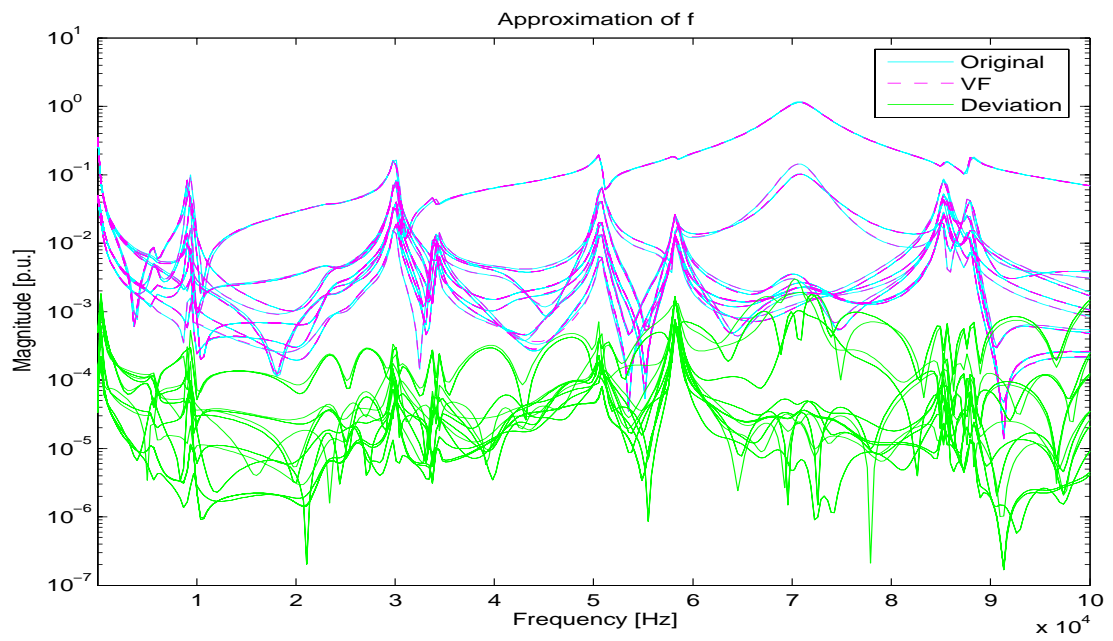
Fig. 6.7 Magnitude plot

ex4d.m – Reducing the likelihood of passivity violations

When a model is to be included as an interacting component in a time domain simulation, passivity is an essential feature in order to guarantee a stable simulation. This requires that the model satisfies the requirement:

$$eig(\text{Re}\{\mathbf{Y}(s)\}) > 0 \;\; \forall s \qquad\qquad (6.1)$$

At infinite frequency this results in that the eigenvalues of **D** must be positive (i.e. **D** is positive definite – P.D.). In addition, the eigenvalues of **E** should be positive (although this is not related to passivity of **Y**). Reference [3] shows that positivity can be enforced by post processing as follows:
1. Set any negative eigenvalues to 0, giving **D**mod, **E**mod
2. Subtract **D**mod +s**E**mod from **Y**, giving **Y**mod.
3. Fit **Y**mod with **D**, **E** enforced to 0. This gives **A**,**B**,**C**
4. The state space model for **Y** is taken as matrices **A**,**B**,**C**,**D**mod,**E**mod

Experience has shown hat large passivity violations tend to occur at frequencies outside the frequency band used in the fitting process, particularly at high frequencies. Reference [4] has suggested to improve on this by throwing out poles that are above the upper frequency limit used in the fitting process.

Example `ex4d.m` shows one way of utilizing these ideas. The procedure is as follows:
1. Fit **Y** in the same way as in `ex4c.m`
2. Throw out all poles at frequencies above the upper band limit
3. Calculate **D**mod, **E**mod, **Y**mod
4. Fit **Y**mod without modifying the poles (use VF.pole_skip=1)
5. passivity checking of resulting model by sweeping eigenvalues of Re{Y(s)} in frequency range 0–2·ω_{max}, at 1200 samples.

The command window dialogue:

```
>> ex4d
Reading data from file ...
-----------------S T A R T-------------------------
****Stacking matrix elements (lower triangle) into single column ...
****Calculating improved initial poles by fitting column sum ...
    Iter 1
    Iter 2
    Iter 3
    Iter 4
    Iter 5
****Fitting column ...
    Iter 1
    Iter 2
    Iter 3
****Throwing out high-frequency poles: ...
```

```
****Refitting residues: ...
****Enforcing positive realness for D, E...
****Refitting C while enforcing D=0, E=0 ...
****Transforming model of lower matrix triangle into state-space
model of full matrix ...
****Generating pole-residue model ...
****Checking passivity (sweeping) ...
-------------------E N D----------------------------
Elapsed time is 19.349000 seconds.
>>
```

The following plots are produced:
- figure(1): Magnitude plot for the fitting of the column **f**(s) and the two subsequent refittings
- figure(2): Phase angles corresponding to figure(1)
- figure(3): Magnitude plot comparing the resulting **Y** after the various stages in the process
- figure(4): Plot comparing eigenvalues of Re{Y(s)}, for the initial fitting results and for the final result.

Beware that this procedure is not a cure-all solution. In particular, discarding high frequency poles can result in that new passivity violations appear in the fitting band. The throwing out of poles can be disabled by setting the parameter remove_Hfpoles to 0 in ex4d.m. Also, the frequency limit for throwing out high frequency poles can be adjusted by parameter factor_HF. For instance, a value 1.1 results in the throwing out of poles above $1.1*\omega_{max}$.
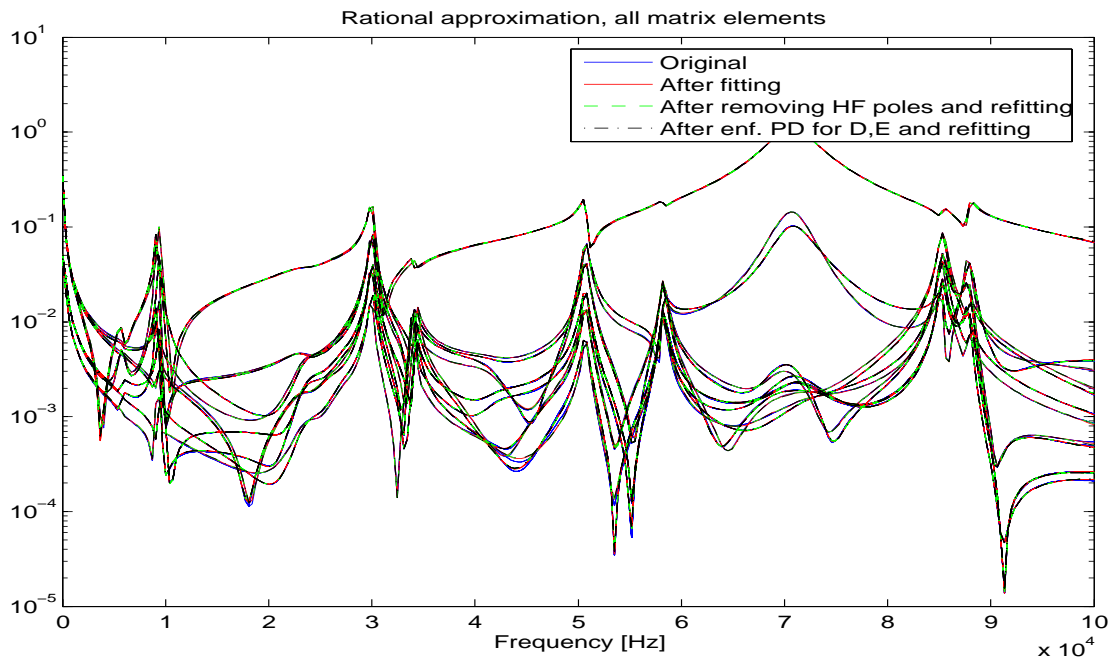

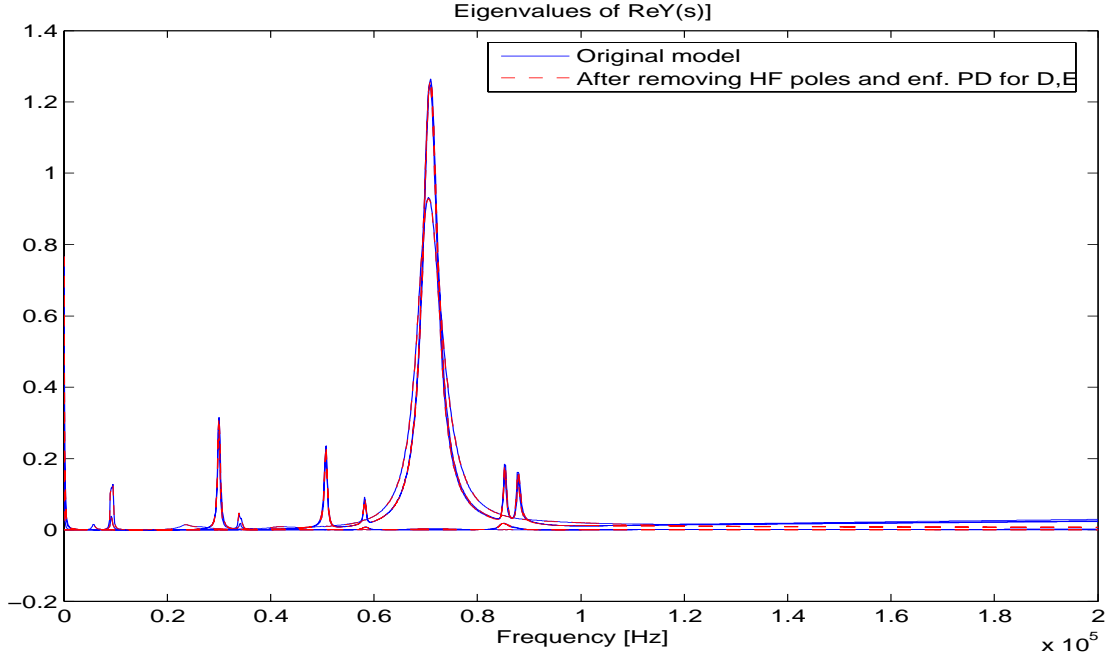Fig. 6.8 Magnitude plots in figure(3)

Fig. 6.9 Eigenvalue plots in figure(4)

Although the the size of passivity violations can be significantly reduced by this approach, violations may remain. Fig. 6.10 shows a closeup of the eigenvalues in Fig. 6.11. It can be seen that some very small violations are remaining (negative eigenvalues).
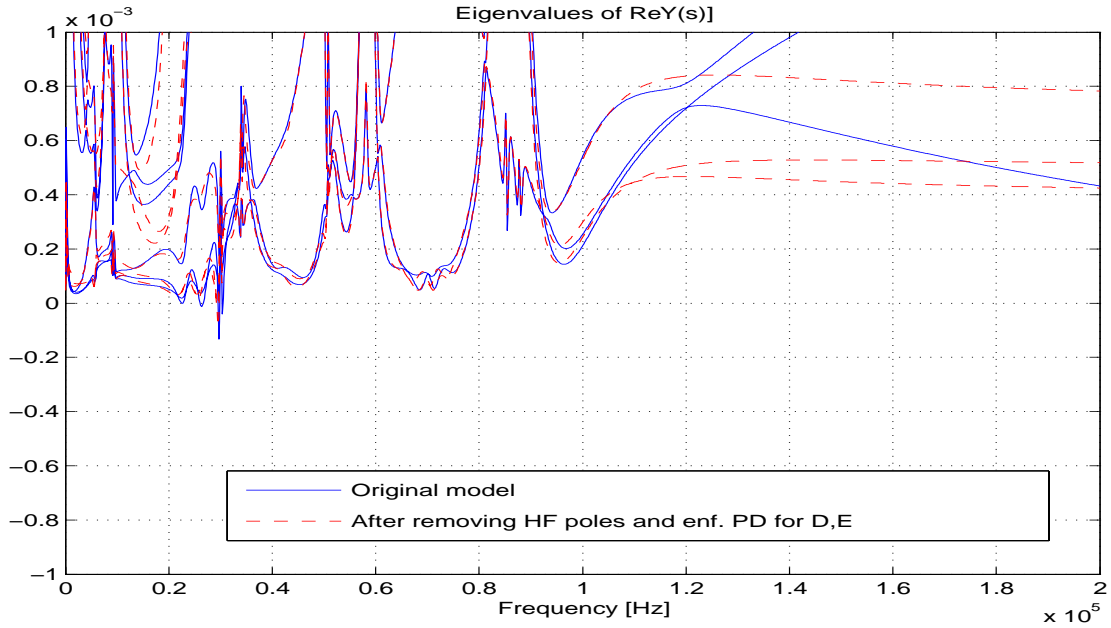

Fig. 6.10 Expanded view of eigenvalue plots

If remaining violations are considered a problem (for instance due to an unstable simulation), such violations can be removed using the approach in [3] which has been implemented in routine QPpassive.m, available in package QPpassive.zip.

Currently, QPpassive.m can only be applied to pole-residue models, meaning that the model must have been obtained by fitting all matrix elements by a common pole set, followed by a conversion using ss2pr.m (as is done in examples ex4c.m and ex4d.m).

ex5.m – Transmission line modelling

- The propagation function **H**(s) of a 5-conductor overhead line is read from file (AC line in parallel with DC line). The lossless time delay has been factored out.
- One column of **H** is fitted with N=14 poles (7 complex pairs) directly in the phase domain (no modal fitting)
- 5 iterations
- No weighting
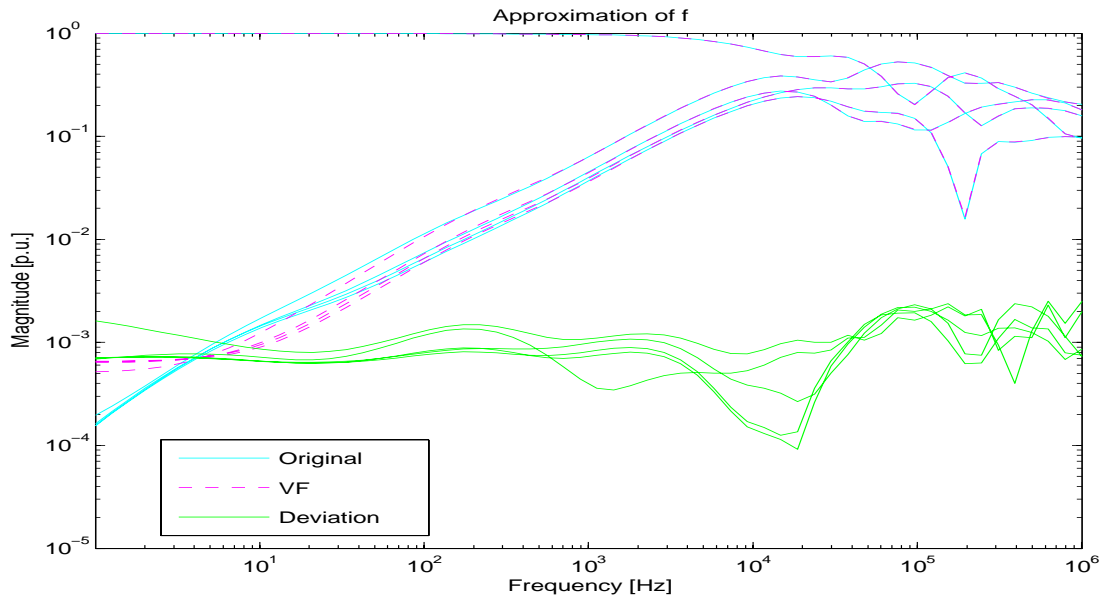
The result is shown in Fig. 6.11 (Magnitude function)

Fig. 6.11 Fitting a column of the propagation function. N=14 poles

It is seen that the "error level" is quite "flat" as function of frequency. Fig. 6.12 shows the fitting result when introducing inverse magnitude weighting.
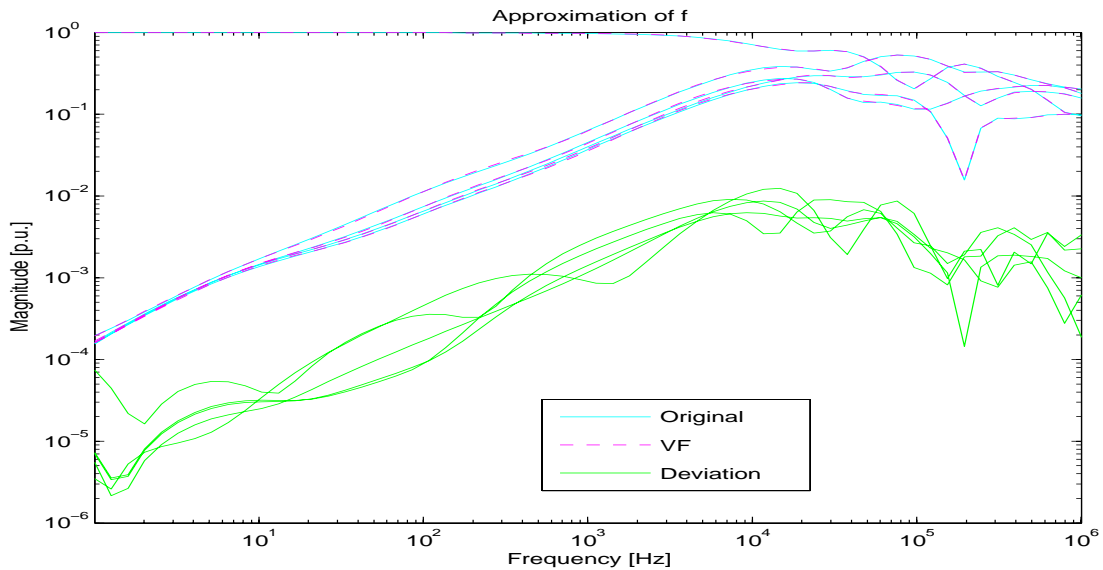
```
weight=1./abs(f);
```


Fig. 6.12 Effect of weighting with inverse of magnitude

We continue with the same example, but remove the inverse weighting (i.e. weight=1 for all frequencies). Fig. 6.13 shows the result when specifying VF.relax=0, resulting in that the relaxation is removed. Comparison with Fig. 6.11 (VF.relax=1) shows that the fitting error is now much higher in the high-frequency region. For an explanation is referred to [2].

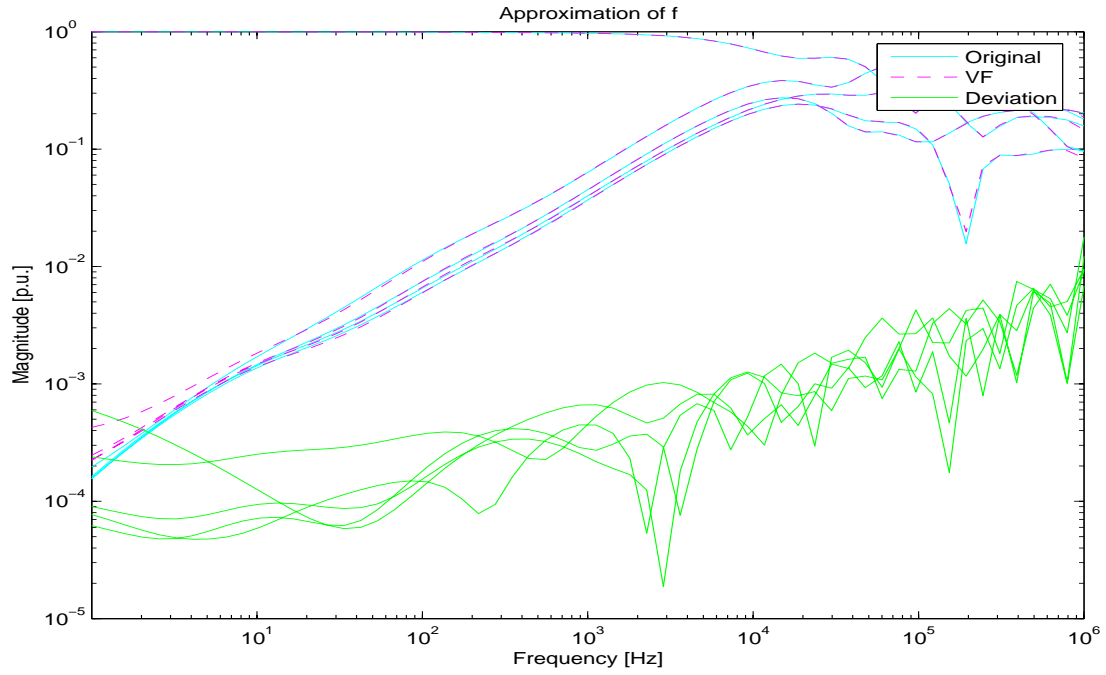Fig. 6.13 Usage with VF.relax=0. No weighting

ex6.m – Fitting a time delay

Time delays naturally occur in systems involving transmission lines. In transmission line modelling by the method of characteristics, the delay effects are easily factored out and removed. However, such factoring cannot easily be achieved when macromodelling a system containing transmission lines. In the following we demonstrate that an uncompensated time delay does not represent an obstacle to VF (although it results in a waste of poles).

- The time delay function $f(s)=\exp(-s\tau)$ is calculated in the frequency range 0–1 MHz with $\tau=10$ µs.
- $f(s)$ is fitted with N=30 poles (15 linearly spaced complex pairs)
- 3 iterations
- No weighting

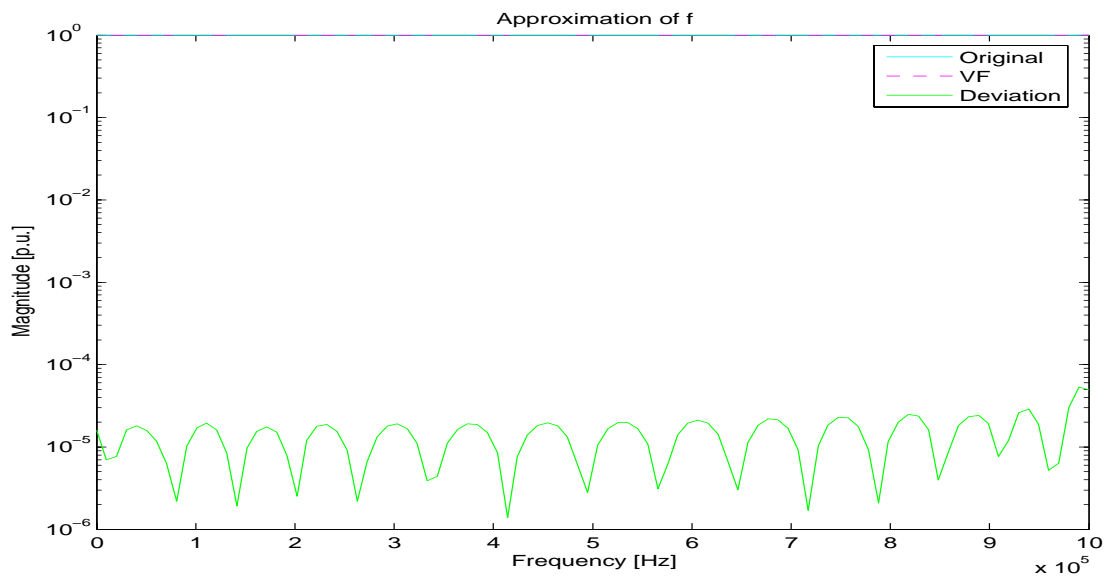The result is shown in Figs 6.14 (magnitude function) and 6.15 (phase angle).
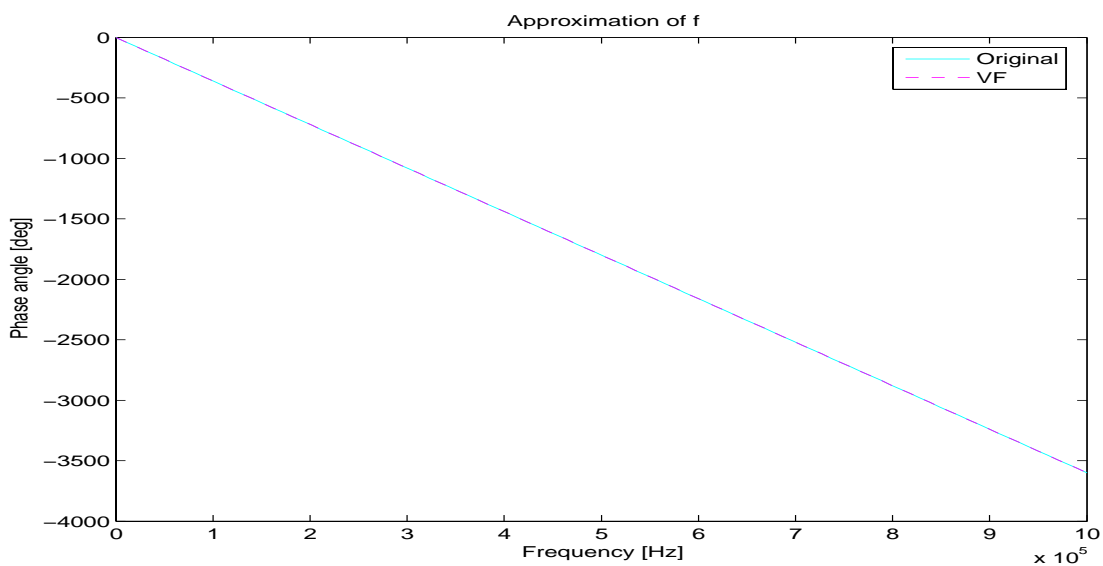

Fig. 6.14 Magnitude plot

Fig. 6.15 Phase plot

7. REFERENCES

[1]   B. Gustavsen and A. Semlyen, "Rational approximation of frequency domain responses by Vector Fitting", *IEEE Trans. Power Delivery, vol. 14, no. 3, pp. 1052-1061, July 1999.*

[2]   B. Gustavsen, "Improving the pole relocating properties of vector fitting", *IEEE Trans. Power Delivery,* accepted.

[3]   B. Gustavsen and A. Semlyen, "Enforcing passivity for admittance matrices approximated by rational functions", *IEEE Trans. Power Systems*, vol. 16, no. 1, pp. 97-104, February 2001.

[4]   D. Saraswat, R. Achar and M.S. Nakhla, "A fast algorithm and practical considerations for passive macromodeling of measured/simulated data", *IEEE Trans. Avdanced Packaging*, vol. 27, no. 1, pp. 57-70, February 2004.